SUNNY-CP: a Portfolio Solver for Constraint Programming

Roberto Amadini[1], Maurizio Gabbrielli[2], and Jacopo Mauro[3]

[1] Department of Computing and Information Systems, University of Melbourne, Australia.
[2] DISI, University of Bologna, Italy / FOCUS Research Team, INRIA, France.
[3] Department of Informatics, University of Oslo, Norway.

Abstract. In Constraint Programming (CP) a portfolio solver combines a variety of different constraint solvers for solving a given problem. This fairly recent approach enables to significantly boost the performance of single solvers, especially when multicore architectures are exploited. In this work we give a brief overview of the portfolio solver `sunny-cp`, and we discuss its performance in the last MiniZinc Challenge —the annual international competition for CP solvers— where it won a gold medal.

1 Introduction

In *Constraint Programming* (CP) the goal is to model and solve Constraint Satisfaction Problems (CSPs) as well as Constraint Optimization Problems (COPs) [21]. Solving a CSP means finding a solution that satisfies all the constraints of the problem, while for COPs the goal is to find a solution that satisfies all the constraints and minimizes (or maximizes) an objective function. A fairly recent trend to solve these problems, based on the well-known *Algorithm Selection* problem [20], consists in building portfolio solvers [11] to boost the performance of existing individual solvers. A *portfolio solver* is a meta-solver that exploits a collection of $n > 1$ constituent solvers $s_1, \ldots, s_n$. When a new, unseen problem p comes, the portfolio solver seeks to predict and run its best solver(s) $s_{i_1}, \ldots, s_{i_k}$ (with $1 \leq k \leq n$) for solving p.

Unfortunately, despite the literature presents plenty of Algorithm Selection approaches [14], a small number of portfolio solvers have been actually developed and adopted in real-life applications [7]. In particular, only few portfolio solvers participated in CP solvers competitions. The first one (for solving CSPs only) was CPHydra [19] that in 2008 won the International CSP Solver Competition. In 2013 a portfolio solver based on Numberjack [12] attended the *MiniZinc Challenge* (MZC) [23], nowadays the only surviving international competition for CP solvers.[4] `sunny-cp` was instead the only portfolio solver that attended the MZCs 2014 and 2015. The first, sequential version of `sunny-cp` [6] had respectable results in the MZC 2014 but remained off the podium. In MZC 2015

[4] The International CSP Solver Competition ended in 2009.

its enhanced, parallel version [4] demonstrated its effectiveness by winning the gold medal in the Open Track of the challenge.

In this paper we extend and integrate the works in [4, 6] by discussing the performance that `sunny-cp` achieved in the MiniZinc Challenges 2014/2015 and by providing some directions for further improvements. The main messages of this paper are: *(i)* a portfolio solver can be better than a single, state-of-the-art CP solver even in scenarios —such as, e.g, the MiniZinc challenge— characterized by relatively small test benchmarks; *(ii)* when a multicore setting is available, a parallel portfolio of sequential CP solvers appears to be more fruitful than a single, parallel CP solver; *(iii)* `sunny-cp` can be a useful baseline to improve a still immature state-of-the-art for the CP field.

2 SUNNY and SUNNY-CP

`sunny-cp` is an open-source CP portfolio solver.[5] Its first implementation was sequential [6], while the current version exploits multicore architectures to run more solvers in parallel and to enable their cooperation via bounds sharing and restarting policies. To the best of our knowledge, it is currently the only parallel portfolio solver able to solve generic CP problems encoded in MiniZinc language [18]. In this section we briefly recall how it works, referring the interested reader to [4] for a more detailed explanation.

`sunny-cp` is built on top of SUNNY algorithm [3]. Given a training set of known problems, a solving timeout T and a portfolio of solvers Π, SUNNY uses the *k-Nearest Neighbours* (k-NN) algorithm to produce a sequential schedule $\sigma = [(s_1, t_1), \ldots, (s_k, t_n)]$ where solver $s_i \in \Pi$ has to be run for t_i seconds and $\sum_{i=1}^{n} t_i = T$. The time slots t_i and the ordering of the solvers s_i are defined according to the average performance of its constituent solvers on the k training instances closer to the problem to be solved. For each problem p, a *feature vector* (i.e., a collection of numerical attributes such as statistics over the variables or the constraints of p) is computed and the Euclidean distance is used to retrieve the k instances in the training set closer to p. For more details about SUNNY features, please see [2]. The sequential schedule σ is then parallelized on the $c \geq 1$ available cores by running the first and most promising $c - 1$ solvers in the k-neighbourhood on the first $c-1$ cores, while the remaining solvers (if any) are assigned to the last available core by linearly widening their allocated times to cover the whole time window $[0, T]$. Note that, as for the SUNNY algorithm, the notion of "promising" is context-sensitive. For CSPs, the performance is measured in terms of number of solved instances and average solving time. For COPs, more general metrics for taking into account also the solutions quality are considered [8]. If there are less solvers than cores, we simply allocate a solver per core.

Solvers are run in parallel and a *"bound-and-restart"* mechanism is used for enabling the bounds sharing between the running COP solvers. This allows to

[5] Publicly available at `https://github.com/CP-Unibo/sunny-cp`.

use the (sub-optimal) solutions found by a solver to narrow the search space of the other scheduled solvers. Since `sunny-cp` treats solvers as black boxes, it does not support the knowledge sharing without their interruption. Therefore, a restarting threshold T_r is used to decide when to stop a solver and restart it with a new bound. A running solver is stopped and restarted if it has not found a solution in the last T_r seconds and its current best bound is obsolete w.r.t. the overall best bound found by another scheduled solver.

`sunny-cp` exploits different state-of-the-art solvers, disparate in their nature (viz., Chuffed, CPX, G12/CBC, G12/FD, G12/LazyFD, G12/Gurobi, Gecode, MinisatID, Choco, HaifaCSP, iZplus, and OR-Tools). It supports several user options which allow to configure the parameters of the SUNNY algorithm (e.g., the number k of neighbours, the timeout T) as well as those of its solvers (e.g., the restarting threshold T_r, the search strategy).

3 SUNNY-CP and the MiniZinc Challenge

The MiniZinc Challenge (MZC) [23] is the annual international competition for evaluating CP solvers. Portfolio solvers can compete in the "Open" class of MZC, which include all the possible CP solvers. In this category, solvers are free to use multiple threads or cores to solve a problem. The scoring system of the MZC is based on a *Borda* count [10] where the performance of solver s on problem p is compared to the performance of each other solver s' on p. If s gives a better answer than s' then it scores 1 point, if it gives a worse solution it scores 0 points. If s and s' give indistinguishable answers the scoring is based on the solving time. Each solver has to solve 100 problem instances —belonging to different classes— defined in the MiniZinc language [18].[6]

Table 1 summarizes the Open class results in the MZCs 2013–2015. The first portfolio solver that attended a MiniZinc Challenge in 2013 was based on Numberjack platform [13]. In the following years, as detailed below, `sunny-cp` was the only portfolio solver which attended the challenge.

MiniZinc Challenge 2014. In 2014 `sunny-cp` was a sequential solver running just one solver at time. We will denote it with `sunny-cp-seq` to distinguish such version from the current, parallel one. `sunny-cp-seq` came with two versions: the default one and a version with pre-solving denoted in Table 1 as `sunny-cp-seq-pre`. The latter added to the default one a static selection of solvers to be run for a short time, before executing the default version of SUNNY in the remaining time. For more details, we refer the reader to [6].

`sunny-cp-seq` obtained respectable results: the two variants ranked 4^{th} and 7^{th} out of 18. Please note that `sunny-cp-seq` had to compete also with parallel solvers, and that 6 of its 8 solvers adopted the "fixed" strategy, i.e., they used the search heuristic defined in the problems instead of using their preferred strategy. To give a measure of comparison, `sunny-cp-seq` in the "Fixed" category —where sequential solvers must follow the search heuristic defined in the model—

[6] Please refer to `http://www.minizinc.org/challenge.html` for further details.

Table 1. Open class results in the MiniZinc Challenges 2013–2015. Portfolio solvers are in bold font, while parallel solvers are marked with *.

2013		2014		2015		
Solver	Score	Solver	Score	Solver	Score	Incomplete
OR-Tools *	1098.85	Chuffed	1326.02	**sunny-cp** *	**1351.13**	**1175.2**
Chuffed	1034.81	OR-Tools *	1086.97	Chuffed	1342.37	1118.16
Choco *	973.27	Opturion CPX	1081.02	**sunny-cp$^-$** *	**1221.88**	**1156.25**
Opturion CPX	929.76	**sunny-cp-seq-pre**	**1066.46**	OR-Tools *	1111.83	1071.67
Gecode *	858.24	Choco *	1007.61	Opturion CPX	1094.09	1036.65
iZplus	758.47	iZplus *	996.32	Gecode *	1049.49	979.05
G12/LazyFD	664.44	**sunny-cp-seq**	**968.64**	Choco *	1027.65	989
Mistral	614.62	G12/LazyFD	784.28	iZplus *	1021.13	1082.92
MZN/Gurobi	589.38	HaifaCSP	781.72	JaCoP	914.97	865.64
JaCoP	577.08	Gecode *	721.48	Mistral *	872.35	878.53
Fzn2smt	556.94	SICStus Prolog	710.51	MinisatID	835.01	793.74
Gecoxicals	512.73	Mistral	705.56	MZN/CPLEX *	799.92	686.64
MZN/CPLEX	447	MinisatID	588.74	MZN/Gurobi *	774.3	697.12
G12/FD	426.53	Picat SAT	588.06	Picat SAT	744.53	626.61
Numberjack *	**383.18**	JaCoP	550.74	MinisatID-MP	637.14	700.35
Picat	363.02	G12/FD	528.26	G12/FD	629.94	664.79
G12/CBC	118.69	Picat CP	404.88	Picat CP	617.22	654.81
		Concrete	353.74	Concrete	533.42	657.2
				YACS *	404.01	553.51
				OscaR/CBLS	403.61	536.17

would have been ranked 1st and 3rd. Moreover, unlike other competitors, the results of `sunny-cp-seq` were computed by considering not only the solving time but also the conversion times from MiniZinc to FlatZinc —the low level specification language used in the MZC by the constituent solvers. This penalized `sunny-cp-seq` especially for the easier instances.

MiniZinc Challenge 2015. Several enhancements of `sunny-cp-seq` were implemented after MZC 2014. `sunny-cp` became parallel, enabling the simultaneous execution of its solvers while retaining the bounds communication for COPs. New state-of-the-art solvers were incorporated in its portfolio. `sunny-cp` became also more stable, usable, configurable and flexible. These improvements, reported in detail in [4] where `sunny-cp` have been tested on large benchmarks of problems, have been reflected in the MZC 2015. `sunny-cp` participated in the competition with two versions: a default one and an "eligible" one, denoted `sunny-cp`$^-$ in Table 1. The difference is that `sunny-cp`$^-$ did not include solvers developed by the organizers, and therefore was eligible for prizes. In particular, `sunny-cp`$^-$ used Choco, Gecode, HaifaCSP, iZplus, MinisatID, Opturion CPX and OR-Tools solvers, while `sunny-cp` used also Chuffed, MZN/Gurobi, G12/FD and G12/LazyFD. Since the availability of eight logical cores, `sunny-cp` performed algorithm selection for computing and distributing the SUNNY sequential schedule, while `sunny-cp`$^-$ launched all its solvers in parallel.

Table 1 shows that `sunny-cp` is the overall best solver while `sunny-cp`$^-$ won the gold medal since Chuffed was not eligible for prizes. The column "Incomplete" of Table 1 refers to the MZC score computed without giving any point for proving optimality or infeasibility. This score, meant to evaluate local search solvers, only takes into account the quality of a solution. With this metric, also

`sunny-cp`$^-$ overcomes Chuffed without having it in the portfolio. There are many reasons behind the success of `sunny-cp` in MZC 2015. Surely the parallelization on multiple cores of state-of-the-art solvers was decisive. We remark that the simultaneous execution of solvers is also cooperative: a running solver can exploit the current best bound found by another one through a proper restart mechanism. Moreover, differently from the MZC 2014, all the solvers where run by using their free version instead of the fixed one. Furthermore, the MZC rules are now less penalizing for portfolio solvers: for the first time, in MZC 2015 the total solving time includes also the conversion time from MiniZinc specifications to FlatZinc.

We underline that the constituent solvers of `sunny-cp` do not exploit multithreading. Hence, the parallel solvers marked with * in Table 1 are not the constituent solvers of `sunny-cp` but their (new) parallel variants. Moreover, we can not be sure whether the sequential solvers we included in the portfolio are exactly the same versions that competed in the competition.

The overall best single solver is Chuffed, which is sequential. Having it in the portfolio is clearly a great benefit for `sunny-cp`. However, even without Chuffed, `sunny-cp`$^-$ is able to provide solutions of high quality (see the "Incomplete" column of Table 1) proving that also the other solvers are important for the success of `sunny-cp`. We would like to remark that, as pointed out in [6], the Borda count used in MZC favors the single best solver for a given problem rather than a portfolio including it. Indeed, compared to the best solver for the given problem, the portfolio solver always has additional overheads (e.g., due to feature extraction or memory contention issues) that penalizes its score.

In 2015, the MZC benchmark used to evaluate the solvers was constituted by 20 different problem classes, each of which consisting of 5 instances: in total, 100 different problem instances (10 CSPs and 90 COPs). `sunny-cp` was the best solver for only two classes: the Capacitated Vehicle Routing problem and the Pizza Voucher problem. This means that the winning performance of `sunny-cp` was due to the good performance it had on-average over all the problem instances. We underline that this paper does not present a novel approach of Algorithm Selection. The interested reader can find a comparison of SUNNY with other selection approaches in [1, 3, 5, 16].

Interestingly, for the Radiation problem class, `sunny-cp`$^-$ scored 0 points because it always provided an unsound answer due to a bug in one of its constituent solvers. This is a sensitive issue that should not be overlooked. On the one hand, a buggy solver inevitably affects the whole portfolio making it buggy as well. On the other hand, not using an unstable solver may penalize the global performance since experimental solvers like Chuffed and iZplus can be very efficient even if not yet in a stable stage. Note that, unlike SAT but similarly to SMT field, most CP solvers are not fully reliable (e.g., in MZC 2014 one third of the solvers provided at least an unsound answer).

When unreliable solvers are used, a possible way to mitigate the problem is to verify *a posteriori* the solution. For instance, another constituent solver can be used for double-checking a solution. Obviously, checking all the solu-

tions of all the solvers implies a slowdown in the solving time. In MZC 2015 `sunny-cp` checked all the solutions of a solver that we already knew to be buggy before the beginning of the challenge. This allowed `sunny-cp` to detect 21 incorrect answers. Without this check its performance would have been dramatically worse: `sunny-cp` would have scored 87.5 points less —thus resulting worse than Chuffed— while `sunny-cp`$^-$ would have scored 206.84 points less, passing from the gold medal to no medal. However, this check was not enough: due to incorrect answers of other constituent solvers `sunny-cp` provided a wrong answer in 5 cases, while `sunny-cp`$^-$ provided 7 wrong answers.

4 Conclusions

We presented an overview of `sunny-cp`, a fairly recent CP portfolio solver relying on the SUNNY algorithm. In particular, we presented its performance in the MiniZinc Challenge —the annual international competition for CP solvers— where in 2015 `sunny-cp` has been the first portfolio solver to win a (gold) medal. For the future of CP portfolio solvers, it would be interesting having more portfolio competitors to improve the state of the art in this field. For example, the state-of-the-art Algorithm Selection approaches of the ICON Challenge 2015 [15] might be adapted to deal with generic CP problems. The SUNNY algorithm itself, which is competitive in the CP scenarios of [3, 5][7], provided very poor performance in the SAT scenarios of the ICON Challenge and [16] shows that it can be strongly improved with a proper training phase.

`sunny-cp` runs in parallel different single-threaded solvers. This choice so far has proved to be more fruitful than parallelizing the search of a single solver. However, the possibility of using multi-threaded solvers may have some benefits when solving hard problems as shown in [17] for SAT problems. The multi-threaded execution also allows search splitting strategies. It is not clear to us if the use of all the available cores, as done by `sunny-cp`, is the best possible strategy. As shown in [22] it is possible that running in parallel all the solvers on the same multicore machine slows down the execution of the individual solvers. Therefore, it may be more convenient to leave free one or more cores and run just the most promising solvers. Unfortunately, it is hard to extrapolate a general pattern to understand the interplay between the solvers and their resource consumption.

Finally, one more direction for further investigations concerns how to deal with unstable solvers. Under these circumstances it is important to find a tradeoff between reliability and performance. Developing an automated way of checking a CP solver outcome when the answer is "unsatisfiable problem" or "optimal solution" is not a trivial challenge: we can not merely do a solution check, but we have to extract and check the actual *explanation* for which the solver provided such an outcome.

[7] We submitted such scenarios, namely `CSP-MZN-2013` and `COP-MZN-2013`, to the Algorithm Selection Library [9].

References

1. Amadini, R., Biselli, F., Gabbrielli, M., Liu, T., Mauro, J.: SUNNY for algorithm selection: a preliminary study. In: CILC, *CEUR Workshop Proceedings*, vol. 1459, pp. 202–206. CEUR-WS.org (2015)
2. Amadini, R., Gabbrielli, M., Mauro, J.: An Enhanced Features Extractor for a Portfolio of Constraint Solvers. In: SAC, pp. 1357–1359. ACM (2014)
3. Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY: a Lazy Portfolio Approach for Constraint Solving. TPLP **14**(4-5), 509–524 (2014)
4. Amadini, R., Gabbrielli, M., Mauro, J.: A multicore tool for constraint solving. In: IJCAI, pp. 232–238. AAAI Press (2015)
5. Amadini, R., Gabbrielli, M., Mauro, J.: Portfolio approaches for constraint optimization problems. AMAI pp. 1–18 (2015)
6. Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY-CP: a sequential CP portfolio solver. In: SAC, pp. 1861–1867. ACM (2015)
7. Amadini, R., Gabbrielli, M., Mauro, J.: Why CP portfolio solvers are (under)utilized? issues and challenges. In: LOPSTR, *LNCS*, vol. 9527, pp. 349–364. Springer (2015)
8. Amadini, R., Stuckey, P.J.: Sequential Time Splitting and Bounds Communication for a Portfolio of Optimization Solvers. In: CP, *LNCS*, vol. 8656, pp. 108–124. Springer (2014)
9. Algorithm Selection Library - coseal. `https://code.google.com/p/coseal/wiki/AlgorithmSelectionLibrary`
10. Chevaleyre, Y., Endriss, U., Lang, J., Maudet, N.: A Short Introduction to Computational Social Choice. In: SOFSEM, *LNCS*, vol. 4362, pp. 51–69. Springer (2007)
11. Gomes, C.P., Selman, B.: Algorithm portfolios. Artif. Intell. **126**(1-2), 43–62 (2001)
12. Hebrard, E., O'Mahony, E., O'Sullivan, B.: Constraint Programming and Combinatorial Optimisation in Numberjack. In: CPAIOR, *LNCS*, vol. 6140, pp. 181–185. Springer-Verlag (2010)
13. Hebrard, E., O'Mahony, E., O'Sullivan, B.: Constraint Programming and Combinatorial Optimisation in Numberjack. In: CPAIOR-10, *LNCS*, vol. 6140, pp. 181–185. Springer-Verlag (2010)
14. Kotthoff, L.: Algorithm Selection for Combinatorial Search Problems: A Survey. AI Magazine **35**(3), 48–60 (2014)
15. Kotthoff, L.: ICON challenge on algorithm selection. CoRR **abs/1511.04326** (2015). URL `http://arxiv.org/abs/1511.04326`
16. Lindauer, M., Bergdoll, R.D., , Hutter, F.: An empirical study of per-instance algorithm scheduling. In: LION (2016)
17. Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Parallel SAT Solver Selection and Scheduling. In: CP, vol. 7514, pp. 512–526. Springer (2012)
18. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a Standard CP Modelling Language. In: CP, *LNCS*, vol. 4741, pp. 529–543. Springer (2007)
19. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. AICS 08 (2009)
20. Rice, J.R.: The Algorithm Selection Problem. Advances in Computers **15**, 65–118 (1976)
21. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)
22. Sabharwal, A., Samulowitz, H.: Insights into Parallelism with Intensive Knowledge Sharing. In: CP, vol. 8656, pp. 655–671. Springer (2014)
23. Stuckey, P.J., Feydy, T., Schutt, A., Tack, G., Fischer, J.: The minizinc challenge 2008-2013. AI Magazine **35**(2), 55–60 (2014)